<u>Report</u>

The first thing we need to see is Graphalgo.cpp where I generated random graph using random numbers by choosing a random number in range of 1000 and checking whether is it greater than 500 or not This gives more random results compare to rand()%2 so I implemented it and the other most important of the graph is that it would return -1 for self in adjacency matrix well the reason for this is It is also standard way of doing and also It will help function I wrote to avoid one of argument as input which I will discuss in a minute

After getting the graph form The input file I created a matrix called colouring matrix which basically stores the colours of the every vertex and initialized to -1 except for first vertex and I already assigned color 0 to the vertex 1 and using random numbers I pushed the vertices to the threads so That partitions remains purely random

Then every threads goes to the colour_part function of the program In colour part first every threads figures out which of its vertex is interior or exterior vertex this can be done by checking whether all its neighbours belongs to this thread or not by this we can find the interior and exterior vertices.The interior vertices are there after coloured based on the colour function which basically finds for colour which is not available for all its neighbours and assign the colour to it

When comes to exterior here comes the difference approaches

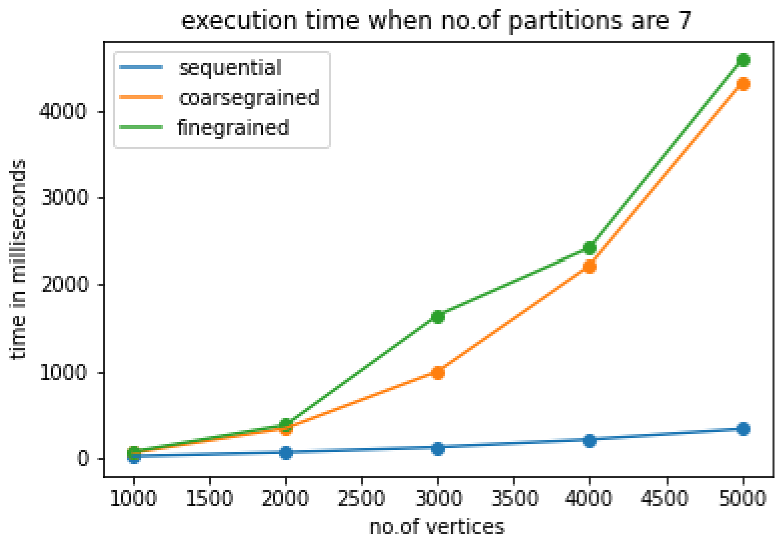
1. Coarse lock :
   In this approach there is one universal lock for every boundary/exterior vertex so first this thread tries to acquire the mutex lock then colours this using greedy way and then releases the lock
2. Fine lock :

In this approach there is one mutex lock for each vertex we should lock every vertex and then colour them as described in the theory and then coloured

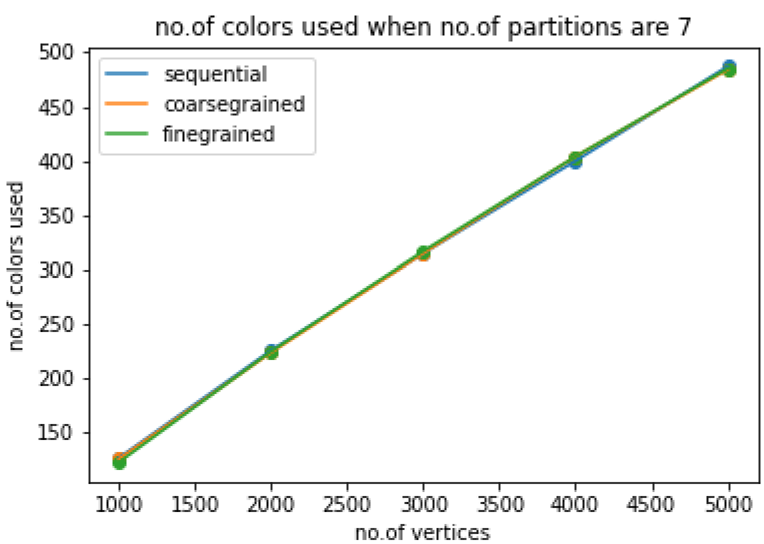
The point which I used / taken it for granted is array[-1]==0 in c++11 standard so whenever there is access of -1 without any safety programming I took is as it is

plot -1 :

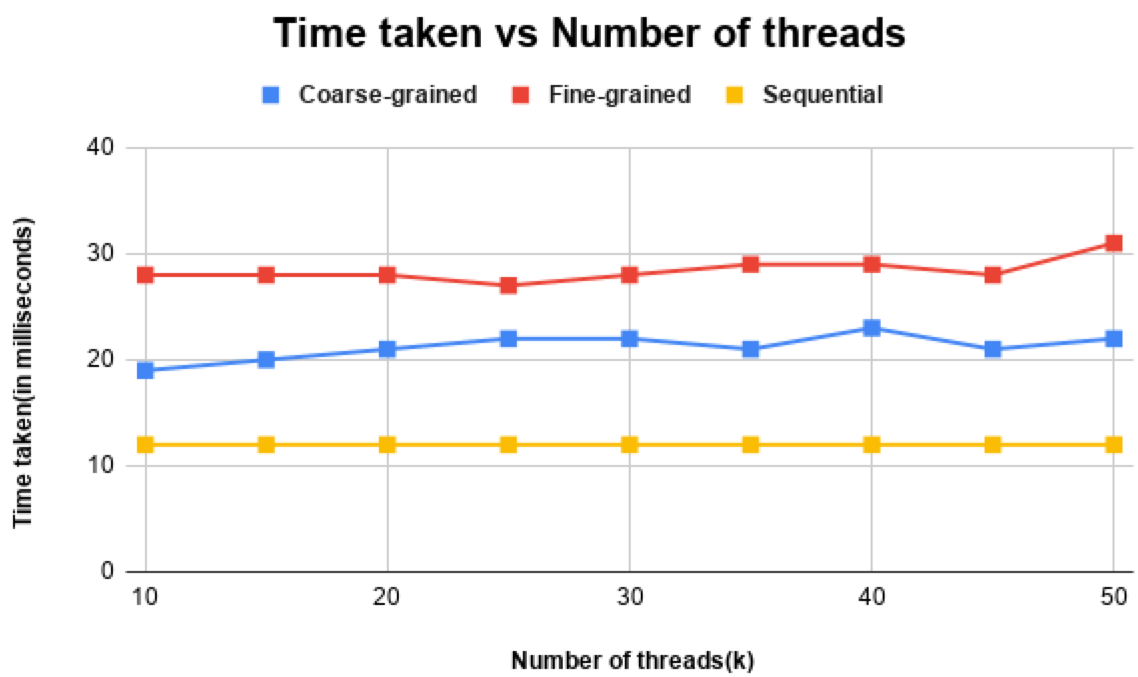
The number of threads are fixed and the input is varying from 1000 to 5000



plot 2 ;
The colours used for the vertices varying from 1000 to 5000

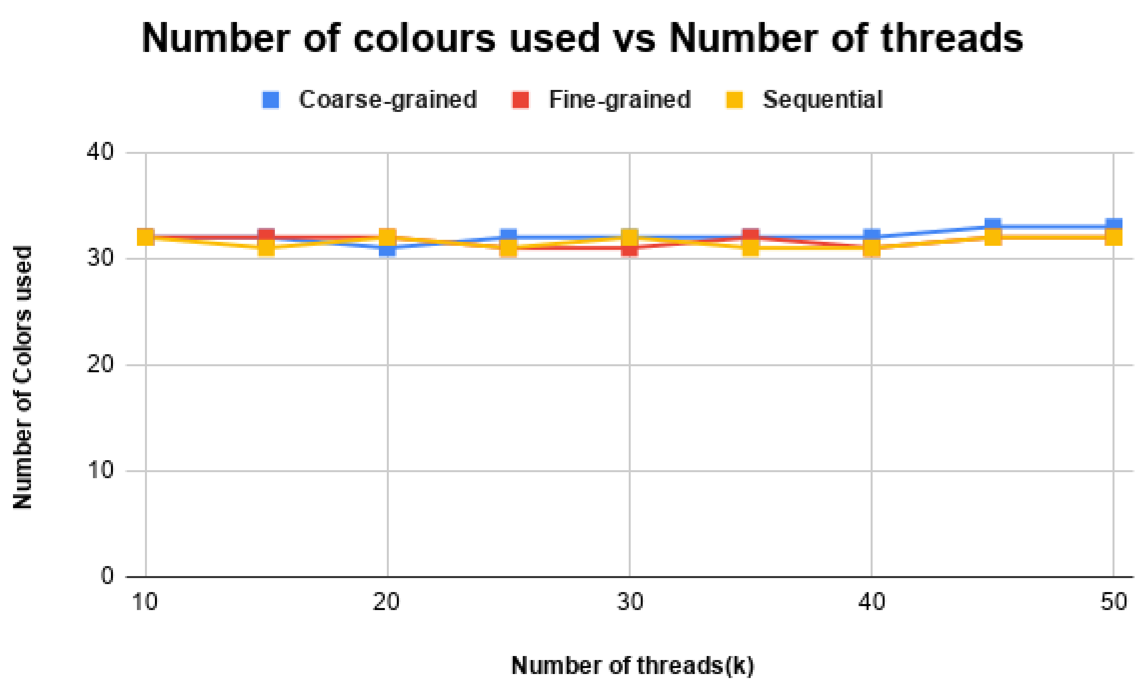


Plot3 :
Varying of time with number of threads



Time taken vs Number of threads
Coarse-grained
Fine-grained
Sequential
Time taken(in milliseconds)
0
10
20
30
40
10
20
30
40
50
Number of threads(k)

Plot4 :



Number of colours used vs Number of threads
Coarse-grained
Fine-grained
Sequential
Number of Colors used
0
10
20
30
40
10
20
30
40
50
Number of threads(k)

From all these graphs this evident that colouring algorithm is very rapid so it doesn’t make much sense for parallelism here and in fact thread creating and merging is also causing overtime for coarse and fine than sequential